

# Multi-Kernel Construction of Polar Codes

Frédéric Gabry, Valerio Bioglio, Ingmar Land, Jean-Claude Belfiore

Mathematical and Algorithmic Sciences Lab

France Research Center, Huawei Technologies Co. Ltd.

Email: {frederic.gabry,valerio.bioglio,ingmar.land,jean.claude.belfiore}@huawei.com

**Abstract**—We propose a generalized construction for binary polar codes based on mixing multiple kernels of different sizes in order to construct polar codes of block lengths that are not only powers of integers. This results in a multi kernel polar code with very good performance while the encoding complexity remains low and the decoding follows the same general structure as for the original Arikan polar codes. The construction provides numerous practical advantages as more code lengths can be achieved without puncturing or shortening. We observe numerically that the error-rate performance of our construction outperforms state-of-the-art constructions using puncturing methods.

**Index Terms**—Polar Codes, Multiple Kernels, Successive Cancellation Decoding.

## I. INTRODUCTION

Polar codes, recently introduced by Arikan in [1], are a new class of channel codes. They are provably capacity-achieving over various classes of channels, and they provide excellent error rate performance for practical code lengths. In their original construction, polar codes are based on the polarization effect of the Kronecker powers of the kernel matrix  $T_2 = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$ , referred to as the binary kernel. The generator matrix of a polar code is a sub-matrix of  $T_2^{\otimes n}$ .

In [1], Arikan conjectured that the polarization phenomenon is not restricted to the powers of the kernel  $T_2$ . This conjecture has been proven in [3], where necessary and sufficient conditions are presented for kernels  $T_l$  with  $l > 2$  to allow for the polarization effect. Recently, researchers proposed polar codes based on larger kernels [4], both linear and non-linear [5], generating polar codes of any block length of the form  $N = l^n$ .

However, many block lengths cannot be expressed in the form  $N = l^n$ . To overcome this limitation, puncturing [6] [7] and shortening [8] techniques have been proposed in the literature. Both shortening and puncturing techniques provide a practical way to construct polar codes of arbitrary lengths based on mother polar codes of length  $N = 2^n$ , albeit with some disadvantages. First, punctured and shortened codes are decoded on the graph of their mother codes, and therefore the decoding complexity can be very high with respect to the shortened code length. Second, puncturing and shortening leads to a substantial loss in terms of polarization speed, and hence a worse error rate performance. Finally, the lack of structure of the frozen sets and puncturing or shortening patterns generated by these methods makes them non-suitable for practical implementation.

In this paper we propose a generalized construction of polar codes based on mixing of kernels of different sizes in order to construct polar codes of different lengths. By using kernels of different dimension in different stages, we show how to construct polar codes of block lengths that are not only powers of integers. With this construction, we obtain a polar code, coined *multi-kernel polar code* in the following, with error-correcting performance comparable or even better than state-of-the-art codes obtained via puncturing/shortening methods. Furthermore, the encoding complexity is similar to binary-kernel polar codes while the decoding follows the same general structure as for usual binary-kernel polar codes. In addition, as for binary-kernel polar codes, the usual decoding algorithms based on list decoding can be used for multi-kernel polar codes, which can as well be enhanced by the use of cyclic redundancy check (CRC) bits [9]. Using this new polar code construction, only a very modest number of bits needs to be punctured or shortened, if at all, and thus the loss in terms of complexity and performance is very small as compared to puncturing or shortening of the original polar codes. In this paper we focus on a construction mixing binary and ternary kernels which allows for code lengths  $N = 2^{n_2} \cdot 3^{n_3}$  without puncturing or shortening.

This paper is organized as follows. In Section II, we present our general construction, including the encoding and decoding, of multi-kernel polar codes. In Section III we describe explicitly the construction for the case of mixed binary and ternary polar codes. In Section IV we illustrate numerically the performance of the codes, and Section V concludes this paper.

## II. MULTI-KERNEL CONSTRUCTION

In this section, we show how to construct polar codes using the proposed multi-kernel construction.

### A. Tanner Graph Construction

In this section, we describe how to construct the Tanner graph for a given transformation matrix  $G_N \triangleq T_{n_1} \otimes \dots \otimes T_{n_s}$  for code length  $N$ . The structure of this graph is a generalization of the Tanner graph of polar codes, shown in Figure 1. Note that the ordering of the  $T_{n_i}$  matrices in the Kronecker product is important, as the Kronecker product is not commutative. The Tanner graph has  $s$  stages, where  $s$  is the number of kernels used to build  $G_N$ . Each stage is constituted by  $B_i = N/n_i$  boxes, where each box is a  $n_i \times n_i$  block given by a  $T_{n_i}$  kernel. As a consequence, stage 1, i.e., the rightmost

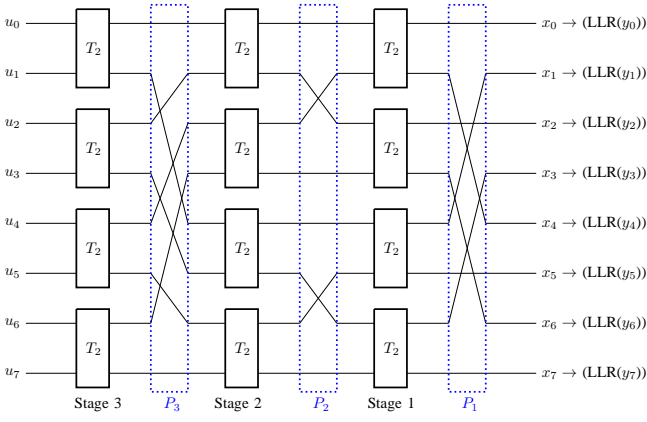


Fig. 1. Tanner graph of the binary-kernel polar code with  $N = 8$  and  $G_8 = T_2 \otimes T_2 \otimes T_2$ .

stage in the graph, is constituted of  $B_1 T_{n_1}$ -boxes and so on, until the last stage  $s$  constituted of  $B_s T_{n_s}$ -boxes.

The connections between stages are generated as follows. We call  $P_i$  the permutation connecting the outputs of boxes of stage  $i - 1$  to the boxes of stage  $i$ . We call  $P_1$  the permutation connecting the encoded bits to the boxes of stage 1. For binary-kernel polar codes,  $P_1$  is given by the bit-reversal permutation. On the other hand, for multi-kernel polar codes,  $P_1$  is the inverse of the product of the other permutations, i.e.,  $P_1 = (P_2 \cdot \dots \cdot P_s)^{-1}$ . The other permutations are based on a basic permutation, which we call  $Q_i$ -canonical permutation. If  $N_i = \prod_{j=1}^{i-1} n_j$  is the partial product of the kernel sizes at stage  $i$ , then  $Q_i$  is a permutation of  $N_{i+1}$  elements defined as in Equation (1) at the top of next page. Now we can write the permutation  $P_i$  as  $P_i = (Q_i | Q_i + N_{i+1} | Q_i + 2N_{i+1} | \dots | Q_i + (N/N_{i+1} - 1)N_{i+1})$ . Note that for the last stage  $P_s = Q_s$ . An example of this construction is showed in Figure 2.

In order to clarify this algorithmic description of the graph construction, let us construct the graph in Figure 1, i.e., for the  $G_8 = T_2 \otimes T_2 \otimes T_2$  transformation matrix, using the proposed algorithm. We first draw 3 stages, each constituted of  $N/2 = 4$   $T_2$  boxes. To construct the edges between Stage 1 and Stage 2, we calculate the canonical permutation  $Q_2 = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 1 & 3 & 2 & 4 \end{pmatrix}$ . Given the canonical permutation, we can calculate  $P_2 = (Q_2 | Q_2 + N_3) = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 1 & 3 & 2 & 4 & 5 & 7 & 6 & 8 \end{pmatrix}$ . We observe indeed, in Figure 1, that bit 1 of stage 1 is connected to bit 1 of stage 2, bit 2 is connected to bit 3, etc. To construct the edges between Stage 2 and Stage 3, we recall that  $P_3 = Q_3$ , and hence  $P_3 = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 1 & 3 & 5 & 7 & 2 & 4 & 6 & 8 \end{pmatrix}$ . Finally,  $P_1 = (P_2 \cdot P_3)^{-1} = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 1 & 5 & 3 & 7 & 2 & 6 & 4 & 8 \end{pmatrix}$ .

### B. Encoding of Multi-Kernel Polar Codes

If the code length is  $N = n_1 \cdot \dots \cdot n_s$ , with  $n_i$  not being necessarily distinct prime numbers, the transformation matrix of the multi-kernel polar code is given as  $G_N = T_{n_1} \otimes \dots \otimes T_{n_s}$ ,

where  $\otimes$  is the Kronecker product. In contrast to binary-kernel polar codes, the ordering of the factors of the Kronecker product is important, since different orderings result in different transformation matrices, with different polarization behaviors. The best kernel ordering for a multi-kernel polar code of length  $N$  and dimension  $K$  can be determined through the density evolution algorithm [2] on the resulting transformation matrix and summing the reliabilities of the  $K$  best bits, or similar methods. In the following, we assume the kernels in the Kronecker product to be already ordered.

Hereafter, the position of the information bits in the length- $N$  message  $u$  is decided according to the reliability of the bits, as for original polar codes. The bit reliabilities can be calculated using the density evolution algorithm or a Monte-Carlo method. Finally, a codeword  $x$  of length  $N$  is obtained as  $x = uG_N$ , where the bits in  $u$  with positions in the frozen set are set to zero. We notice that the encoding may be performed on the Tanner graph of the multi-kernel polar code. In Section III we will give an example of this construction for the case  $N = 6$ .

### C. Decoding of Multi-Kernel Polar Codes

The decoding of multi-kernel polar codes is done similarly to binary-kernel polar codes using successive cancellation (SC) decoding on the Tanner graph of the code. In practice, the log-likelihood ratios (LLRs) are passed along the Tanner graph from the right to the left, while the hard decisions on the decoded bits are passed from the left to the right. Assuming that  $G_N = T_{n_1} \otimes \dots \otimes T_{n_s}$ , the LLRs go through  $B_1 T_{n_1}$ -boxes of size  $n_1 \times n_1$  on the first stage, until the  $B_s T_{n_s}$ -boxes of size  $n_s$ -by- $n_s$  on the last stage. We call  $\text{LLR}(j, i)$  and  $u(j, i)$  the values taken by the LLR and the hard decision of bit  $i$  at stage  $j$  of the Tanner graph respectively. The update of the LLRs is then done according to update functions corresponding to the kernel used at a given stage. The algorithmic description of this section will be clarified in the description of the mixed binary and ternary example in Section III with a detailed description of the update functions for LLRs and hard decisions for kernels  $T_2$  and  $T_3$ .

SC decoding operates as follows. Initially, the LLRs of the coded bits  $x_i$  based on the received vector  $y$  are calculated at the receiver. The received signal is decoded bit-by-bit using LLR propagation through the graph to retrieve the transmitted message  $\mathbf{u} = [u_0, \dots, u_i, \dots, u_{N-1}]$ . For every bit  $u_i$ , the nature of its position  $i$  is initially checked. If  $u_i$  is a frozen bit, it is decoded as  $\hat{u}_i = 0$ , and the decoder moves on to the next bit. If  $u_i$  is an information bit, its LLR is recursively calculated to make a hard decision, starting by  $\text{LLR}(s, i)$ . In general, the calculation of  $\text{LLR}(j, l)$  is done using the  $\text{LLR}(j-1, \cdot)$  and the hard decisions  $u(j, \cdot)$  that are connected to the  $T_{n_j}$  box outputting  $\text{LLR}(j, l)$ . The value of  $\text{LLR}(j, l)$  is calculated according to the update rules corresponding to the  $T_{n_j}$  kernel. These update rules are given in Section III for the case of the binary and ternary kernels used in the proposed implementation of the code. Using such a recursive procedure, the algorithm will arrive to the calculations of  $\text{LLR}(1, \cdot)$  values

$$Q_i = \begin{pmatrix} 1 & 2 & \dots & N_i & N_i + 1 & N_i + 2 & \dots & (n_i - 1)N_i + 1 & \dots & N_{i+1} \\ 1 & n_i + 1 & \dots & (N_i - 1)n_i + 1 & 2 & n_i + 2 & \dots & n_i & \dots & N_{i+1} \end{pmatrix}. \quad (1)$$

in the graph, which are obtainable using the channel LLRs and the updating rules corresponding to  $T_{n_1}$ . Hence all the LLRs needed for the computation of  $\text{LLR}(s, i)$  end up being calculated, and finally  $\text{LLR}(s, i)$  is computed. Finally, the bit is decoded as  $\hat{u}_i$  given by the hard decision corresponding to the sign of  $\text{LLR}(s, i)$ .

### III. EXAMPLE: MIXING BINARY AND TERNARY KERNELS

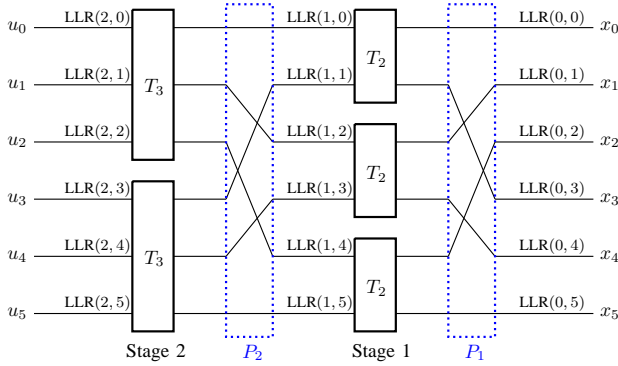
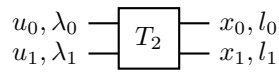


Fig. 2. Tanner graph of the multi-kernel polar code with  $N = 6$  and  $G_6 = T_2 \otimes T_3$ .

In this section, we illustrate the general code construction through a simple example using binary and ternary kernels, which allow us to construct codes for any length expressed as  $N = 2^{l_2} \cdot 3^{l_3}$ . In particular, we will illustrate the encoding and decoding of the proposed codes using a  $N = 6$  code with transformation matrix  $G_6 = T_2 \otimes T_3$ , see Figure 2. We initially describe the updating rules for the binary and ternary kernels used in our implementation. Even if the update rules for the binary kernels are well-known as the canonical component of original polar codes, the update rules for ternary codes are mostly unknown in the literature. Then, we show how to construct the Tanner graph of multi binary and ternary kernel polar codes. Finally, we show how the SC decoding procedure described previously is applied to the Tanner graph.

#### A. Decoding rules and functions for $T_2$ kernels

In this section we remind the well-known decoding rules for binary-kernel polar codes [1]. The fundamental  $T_2$  block can be depicted as



where  $(u_0 \ u_1) \cdot T_2 = (x_0 \ x_1)$ , with  $T_2 = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$ , and where  $\lambda_i$  and  $l_i$  denote the LLRs and  $u_i$  and  $x_i$  denote the hard decisions on the bits. This corresponds to the hard-decision

update rules

$$\begin{aligned} x_0 &= u_0 \oplus u_1, \\ x_1 &= u_1. \end{aligned}$$

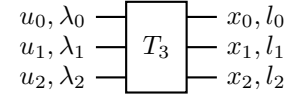
The inverse of the update rules are  $u_0 = x_0 \oplus x_1$  and  $u_1 = x_1 = u_0 \oplus x_0$ , which correspond to the message update equations

$$\begin{aligned} \lambda_0 &= l_0 \boxplus l_1, \\ \lambda_1 &= (-1)^{u_0} \cdot l_0 + l_1, \end{aligned}$$

where  $a \boxplus b \triangleq 2 \tanh^{-1}(\tanh \frac{a}{2} \cdot \tanh \frac{b}{2})$ .

#### B. Decoding rules and functions for $T_3$ kernels

In this section we describe the decoding rules for  $T_3$  kernels. The fundamental  $T_3$  block can be depicted as follows



where  $(u_0 \ u_1 \ u_2) \cdot T_3 = (x_0 \ x_1 \ x_2)$ , with  $T_3 \triangleq \begin{pmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix}$ .

Consequently, the hard-decisions update rules are

$$\begin{aligned} x_0 &= u_0 \oplus u_1, \\ x_1 &= u_0 \oplus u_2, \\ x_2 &= u_0 \oplus u_1 \oplus u_2. \end{aligned}$$

Inverting these equations, we get  $u_0 = x_0 \oplus x_1 \oplus x_2$ ,  $u_1 = u_0 \oplus x_0 = x_1 \oplus x_2$  and  $u_2 = u_0 \oplus x_1 = u_0 \oplus u_1 \oplus x_2$ , from which we get the message update equations

$$\begin{aligned} \lambda_0 &= l_0 \boxplus l_1 \boxplus l_2, \\ \lambda_1 &= (-1)^{u_0} \cdot l_0 + l_1 \boxplus l_2, \\ \lambda_2 &= (-1)^{u_0} \cdot l_1 + (-1)^{u_0 \oplus u_1} \cdot l_2. \end{aligned}$$

#### C. Construction Example for $G_6 = T_2 \otimes T_3$

The Tanner graph of the length-6 code obtained from the  $G_6 = T_2 \otimes T_3$  transformation matrix is shown in Figure 2. Let  $u = [u_0, \dots, u_5]$  be the bits to be encoded,  $K$  of which are information bits and  $6 - K$  are frozen bits (according to the frozen set  $\mathcal{F}$ ). The codeword  $x = [x_0, \dots, x_5]$  is then constructed as  $x = u \cdot G_6$ , or following the hard-decision update rules of the Tanner graph.

The decoding starts with  $u_0$ . If  $u_0$  is a frozen bit, its value  $\hat{u}_0$  is set to 0 and the decoding continues with  $u_1$ . Otherwise,  $\text{LLR}(2, 0)$  is calculated to make a hard decision on the value of  $u_0$ . According to the Tanner graph and the decoding rules of the  $T_3$  kernel,  $\text{LLR}(2, 0) = \text{LLR}(1, 0) \boxplus \text{LLR}(1, 2) \boxplus \text{LLR}(1, 4)$ . The values of these intermediate LLRs have to be calculated. According to the update rules of kernel  $T_2$  and

the Tanner graph of the multi-kernel polar code,  $\text{LLR}(1, 0) = \text{LLR}(0, 0) \boxplus \text{LLR}(0, 3)$ , while  $\text{LLR}(1, 2) = \text{LLR}(0, 1) \boxplus \text{LLR}(0, 4)$  and  $\text{LLR}(1, 4) = \text{LLR}(0, 2) \boxplus \text{LLR}(0, 5)$ . Since LLRs at stage 0 are the LLRs of the received signal, the recursion stops and  $\text{LLR}(2, 0)$  is calculated. We notice that, in order to speed up the decoding, the values of the LLRs and hard-decisions at the intermediate stages can be stored, since they remain constant during the decoding once calculated.

#### IV. NUMERICAL ILLUSTRATIONS

In the following, we show the performance of multi-kernel polar codes for the example described in Section III, where binary and ternary kernels are mixed.

In particular, we show the performance of the multi-kernel polar codes of length  $N = 72 = 2^3 \cdot 3^2$  and  $N = 48 = 2^4 \cdot 3$ . For  $N = 72$  there exist 10 possible permutations of the kernels  $T_2$  and  $T_3$ , i.e., 10 different ways to construct the multi-kernel polar code. For  $K = 36$  information bits, density evolution analysis suggests to use the transformation matrix  $G_{72} = T_3 \otimes T_2 \otimes T_2 \otimes T_2 \otimes T_3$  to build the code. For  $N = 48$ , only 5 permutations of the kernels  $T_2$  and  $T_3$  can be generated, and the transformation matrix  $G_{48} = T_2 \otimes T_2 \otimes T_2 \otimes T_2 \otimes T_3$  has been selected by the density evolution analysis for  $K = 24$ . Hence both codes are designed for a rate equal to  $1/2$ .

In Figure 3 and in Figure 4 we compare the Block Error Rate (BLER) performance of the proposed multi-kernel polar code for an additive white Gaussian noise (AWGN) channel against state-of-the-art punctured and shortened polar codes, proposed in [6] and in [8], respectively. For  $N = 72$  a mother polar code of length 128 is used, while for  $N = 48$  the mother polar code has length 64. In particular, we show the SC-List decoding performance of the codes for list size  $L = 8$  and  $L = 1$ , the latter corresponding to SC decoding.

We observe that for both cases, our construction significantly outperforms state-of-the-art punctured and shortened polar codes. Even more, the decoder of the proposed multi-kernel polar codes has a lower complexity compared to the decoder of the state-of-the-art punctured polar codes, due to the larger length of the mother polar code for that construction.

#### V. CONCLUSIONS

In this paper, we proposed a generalized polar code construction based on the polarization of multiple kernels. The construction provides numerous practical advantages, as more code lengths can be achieved without puncturing or shortening and only modest puncturing and shortening is required to achieve any arbitrary code length. For an example with binary and ternary kernels, we observed numerically that the error-rate performance of our construction clearly outperforms state-of-the-art constructions using puncturing or shortening methods.

#### REFERENCES

[1] E. Arıkan, "Channel polarization: a method for constructing capacity-achieving codes for symmetric binary-input memoryless channels," *IEEE Transactions on Information Theory*, vol. 55, no. 7, pp. 3051–3073, July 2009.

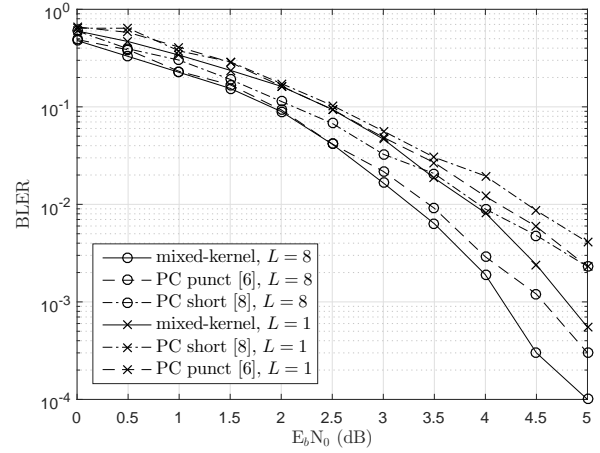


Fig. 3. Performance of length-72 codes with SCL decoding for rate  $1/2$ .

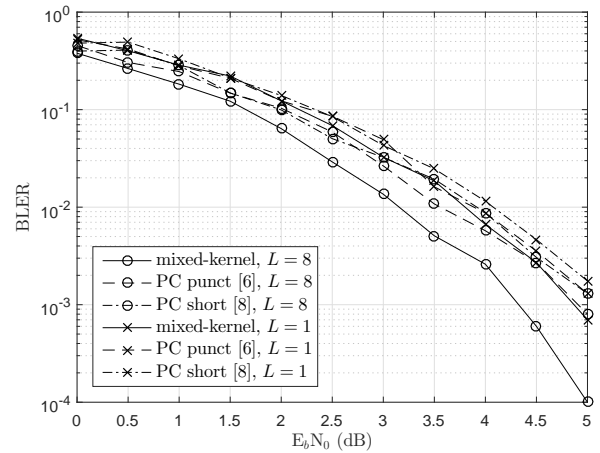


Fig. 4. Performance of length-48 codes with SCL decoding for rate  $1/2$ .

[2] R. Mori and T. Tanaka, "Performance of polar codes with the construction using density evolution," *IEEE Communications Letters*, vol. 13, no. 7, pp. 519–521, July 2009.

[3] S. B. Korada, E. Sasoglu, and R. Urbanke, "Polar codes: Characterization of exponent, bounds, and constructions," *IEEE Transactions on Information Theory*, vol. 56, no. 12, pp. 6253–6264, Dec. 2010.

[4] N. Presman, O. Shapira, S. Litsyn, T. Etzion, and A. Vardy, "Binary polarization kernels from code decompositions," *IEEE Transactions on Information Theory*, vol. 61, no. 5, pp. 2227–2239, May 2015.

[5] H.-P. Lin, S. Lin, and K. Abdel-Ghaffar, "Linear and nonlinear binary kernels of polar codes of small dimensions with maximum exponents," *IEEE Transactions on Information Theory*, vol. 61, no. 10, pp. 5253–5270, Oct. 2015.

[6] K. Niu, K. Chen, and J.-R. Lin, "Beyond turbo codes: Rate-compatible punctured polar codes," in *IEEE International Conference on Communications (ICC)*, Budapest, Hungary, June 2013.

[7] L. Zhang, Z. Zhang, X. Wang, Q. Yu, and Y. Chen, "On the puncturing patterns for punctured polar codes," in *IEEE International Symposium on Information Theory (ISIT)*, Hawaii, U.S.A., July 2014.

[8] R. Wang and R. Liu, "A novel puncturing scheme for polar codes," *IEEE Communications Letters*, vol. 18, no. 12, pp. 2081–2084, Dec. 2014.

[9] I. Tal and A. Vardy, "List decoding of polar codes," in *IEEE International Symposium on Information Theory Proceedings (ISIT)*, St Petersburg, Russia, July 2011.